UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/871,496	05/31/2001	Timothy Gerrit Deboer	CA920010032US1	8687

25259 7590 12/16/2003

IBM CORPORATION
3039 CORNWALLIS RD.
DEPT. T81 / B503, PO BOX 12195
REASEARCH TRIANGLE PARK, NC 27709

EXAMINER	
STARKS, WILBERT L	

ART UNIT	PAPER NUMBER
2121	

DATE MAILED: 12/16/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 09/871,496	Applicant(s) DEBOER ET AL.	
	Examiner Wilbert L. Starks, Jr.	Art Unit 2121	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☐ Responsive to communication(s) filed on *31 May 2001*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☐ Claim(s) *1-19* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☐ Claim(s) *1-19* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. §§ 119 and 120

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

13)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

a) ☐ The translation of the foreign language provisional application has been received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____.
4) ☐ Interview Summary (PTO-413) Paper No(s). _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other:

DETAILED ACTION

Claim Rejections - 35 USC § 101

1. 35 U.S.C. §101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

the invention as disclosed in claims 1-19 is directed to non-statutory subject matter.

2. Claims 1-15 and 17 are not claimed to be practiced on a computer, therefore, it is clear that the claims are not limited to practice in the technological arts. On that basis alone, they are clearly nonstatutory.

3. Regardless of whether any of the claims are in the technological arts, none of them is limited to practical applications in the technological arts. Examiner finds that *In re Warmerdam*, 33 F.3d 1354, 31 USPQ2d 1754 (Fed. Cir. 1994) controls the 35 USC §101 issues on that point for reasons made clear by the Federal Circuit in *AT&T Corp. v. Excel Communications, Inc.*, 50 USPQ2d 1447 (Fed. Cir. 1999). Specifically, the Federal Circuit held that the act of:

> ...[T]aking several abstract ideas and manipulating them together adds nothing to the basic equation. *AT&T v. Excel* at 1453 quoting *In re Warmerdam*, 33 F.3d 1354, 1360 (Fed. Cir. 1994).

Examiner finds that Applicant's "object oriented application component" references are just such abstract ideas.

4. Examiner bases his position upon guidance provided by the Federal Circuit in *In re Warmerdam*, as interpreted by *AT&T v. Excel*. This set of precedents is within the same line of cases as the *Alappat-State Street Bank* decisions and is in complete agreement with those decisions. *Warmerdam* is consistent with *State Street*'s holding that:

> Today we hold that *the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price*, constitutes a practical application of a mathematical algorithm, formula, or calculation because it produces 'a useful, concrete and tangible result" – *a final share price momentarily fixed for recording purposes and even accepted and relied upon by regulatory authorities and in subsequent trades.* (emphasis added) *State Street Bank* at 1601.

5. True enough, that case later eliminated the "business method exception" in order to show that business methods were not per se nonstatutory, but the court clearly *did not* go so far as to make business methods *per se statutory*. A plain reading of the excerpt above shows that the Court was *very specific* in its definition of the new *practical application*. It would have been much easier for the court to say that "business methods were per se statutory" than it was to define the practical application in the case as "...the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price..."

6. The court was being very specific.

7. Additionally, the court was also careful to specify that the "useful, concrete and tangible result" it found was "a final share price momentarily fixed for recording purposes and even accepted and relied upon by regulatory authorities and in subsequent trades." (i.e. the trading activity is the further practical use of the real world monetary data beyond the transformation in the computer – i.e., "post-processing activity".)

8. Applicant cites no such specific results to define a useful, concrete and tangible result. Neither does Applicant specify the associated practical application with the kind of specificity the Federal Circuit used.

9. Furthermore, in the case *In re Warmerdam*, the Federal Circuit held that:

> ...[T]he dispositive issue for assessing compliance with Section 101 in this case is whether the claim is for a process that goes beyond simply manipulating 'abstract ideas' or 'natural phenomena' ... As the Supreme Court has made clear, '[a]n idea of itself is not patentable, ... *taking several abstract ideas and manipulating them together adds nothing to the basic equation*. In re Warmerdam 31 USPQ2d at 1759 (emphasis added).

10. Since the Federal Circuit held in *Warmerdam* that this is the "dispositive issue" when it judged the usefulness, concreteness, and tangibility of the claim limitations in that case, Examiner in the present case views this holding as the dispositive issue for determining whether a claim is "useful, concrete, and tangible" in similar cases. Accordingly, the Examiner finds that Applicant manipulated a set of abstract "object oriented application components" to solve purely algorithmic problems in the abstract (i.e., what *kind* of "information" is used in the application components? Algebraic word problems? Boolean logic problems? Fuzzy logic algorithms? Probabilistic word problems? Philosophical ideas? Even vague expressions, about which even reasonable persons could differ as to their meaning? Combinations thereof?) Clearly, a claim for manipulation of "object oriented application components" is provably even more abstract (and thereby less limited in practical application) than pure "mathematical algorithms" which the Supreme Court has held are per se nonstatutory – in fact, it *includes* the expression of nonstatutory mathematical algorithms.

11. Since the claims are not limited to exclude such abstractions, the broadest reasonable interpretation of the claim limitations includes such abstractions. Therefore, the claims are impermissibly abstract under 35 U.S.C. 101 doctrine.

12. Since *Warmerdam* is within the *Alappat-State Street Bank* line of cases, it takes the same view of "useful, concrete, and tangible" the Federal Circuit applied in *State Street Bank*. Therefore, under *State Street Bank*, this could not be a "useful, concrete and tangible result". There is only manipulation of abstract ideas.

13. The Federal Circuit validated the use of *Warmerdam* in its more recent *AT&T Corp. v. Excel Communications, Inc.* decision. The Court reminded us that:

> Finally, **the decision in In re Warmerdam**, 33 F.3d 1354, 31 USPQ2d 1754 (Fed. Cir. 1994) **is not to the contrary**. *** The court found that the claimed process did nothing more than manipulate basic mathematical constructs and concluded that **'taking several abstract ideas and manipulating them together adds nothing to the basic equation'**; hence, the court held that the claims were properly rejected under §101 ... Whether one agrees with the court's conclusion on the facts, the **holding** of the case is a straightforward application of the basic principle that mere laws of nature, natural phenomena, and abstract ideas are not within the categories of inventions or discoveries that may be patented under §101. (emphasis added) AT&T Corp. v. Excel Communications, Inc., 50 USPQ2d 1447, 1453 (Fed. Cir. 1999).

14. Remember that in *In re Warmerdam*, the Court said that this was the dispositive issue to be considered. In the *AT&T* decision cited above, the Court reaffirms that this is the issue for assessing the "useful, concrete, and tangible" nature of a set of claims under 101 doctrine. Accordingly, Examiner views the *Warmerdam* holding as the dispositive issue in this analogous case.

15. The fact that the invention is merely the manipulation of *abstract ideas* is clear. The data referred to by Applicant's phrase "object oriented application component" is simply an abstract construct that does not limit the claims to the transformation of real world data (such as monetary data or heart rhythm data) by some disclosed process. Consequently, the necessary conclusion under *AT&T, State Street* and *Warmerdam,* is

straightforward and clear. The claims take several abstract ideas (i.e., "object oriented application component" in the abstract) and manipulate them together adding nothing to the basic equation. Claims 1-19 are, thereby, rejected under 35 U.S.C. 101.

16. Regarding the apparent "product of manufacture" claims in claims 16, 18, and 19 the invention is still found to be nonstatutory. Any other finding would be at variance with current case law. Specifically, the Federal Circuit held in *AT&T v. Excel*, 50 USPQ2d 1447 (Fed. Cir. 1999) that:

> Whether stated implicitly or explicitly, we consider the scope of Section 101 to be **the same regardless of the form** -- machine or process -- in which a particular claim is drafted. AT&T v. Excel, 50 USPQ2d 1447, 1452 citing In re Alappat, 33 F.3d at 1581, 31 USPQ2d at 1589 (Rader, J., concurring) (emphasis added.)

17. Examiner considers the scope of Section 101 to be the same regardless of whether Applicant *claims* a "process", "machine", or "product of manufacture". the attempts to limit claims 16, 18, and 19 to "product of manufacture" claims are insufficient by themselves to limit the claims to statutory subject matter. Examiner's position is clearly consistent with *Alappat,* and *AT&T* and is implicitly consistent with *Warmerdam* and *State Street*. Accordingly, those claims are also properly rejected.

Claim Rejections - 35 USC § 112

The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the

> art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claims 1-19 are rejected under 35 USC 112, first paragraph because current case law (and accordingly, the MPEP) require such a rejection if a 101 rejection is given because when Applicant has not in fact disclosed the practical application for the invention, as a matter of law there is no way Applicant could have disclosed *how* to practice the *undisclosed* practical application. This is how the MPEP puts it:

> ("The how to use prong of section 112 **incorporates as a matter of law** the requirement of 35 U.S.C. 101 that the specification disclose as a matter of fact a practical utility for the invention.... If the application fails as a matter of fact to satisfy 35 U.S.C. § 101, then the application **also fails as a matter of law** to enable one of ordinary skill in the art to use the invention under 35 U.S.C. § 112."); In re Kirk, 376 F.2d 936, 942, 153 USPQ 48, 53 (CCPA 1967) ("Necessarily, compliance with § 112 requires a description of how to use presently useful inventions, **otherwise an applicant would anomalously be required to teach how to use a useless invention**.").See, MPEP 2107.01(IV), quoting In re Kirk (emphasis added).

Therefore, claims 1-19 are rejected on this basis.

Claim Rejections - 35 USC § 102

18. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

19. Claims 1- 19 are rejected under 35 U.S.C. 102(e) as being anticipated by Helgeson et al (U.S. Patent Number 6,643,652 B2; dated 04 November 2003; class 707; subclass 010). Specifically:

Claim 1

Claim 1's "providing a client side application portion for presenting a view to a user of a Web browser, said view allowing user interactions with said view, where some of said interactions specify given tests to perform on said component; and" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 1's "executing a server side application portion for receiving indications of said user interactions with said client side application portion and, responsive to said indications, performing said given tests on said component." is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific

> Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 2

Claim 2's "The method of claim 1 wherein said object-oriented application component is an Enterprise JavaBean." is anticipated by Helgeson et al, col. 2, lines 1-6, where it recites:

> ...required to allow distributed objects to communicate with each other across either the client-server or server-server boundary has been provided by the **EnterpriseJavaBeans (EJB).**TM. component architecture.This new architectural system and related tools and systems are well documented and well known to those skilled in these arts.

Claim 3

Claim 3's "The method of claim 2 where a **runtime** execution environment in which said given tests on said Enterprise JavaBean are performed is the same **runtime** execution environment in which said server side application portion is executed." is anticipated by Helgeson et al, col. 11, lines 39-67, where it recites:

> Referring now to FIG. 4, the tier 3 applications server 307 is expanded in FIG. 4 to illustrate the Business Applications Platform 415 of the present invention. In FIG. 4, the Platform contains an Interface Server 417, an Information Server 419, an Interconnect Server 423 and a Business Server 421. All of these Servers 417, 419, 421 and 423 may physically resign on the same hardware platform (such as a UNIX box or a Microsoft.TM. NT.TM. platform), or each server may reside on a separate hardware box, or any

> combination of servers and hardware boxes. Each of the servers may have included a JAVA Virtual Machine.TM. and the related **runtime** support. The electronic communications between these servers may use the XML protocol (409, 425, 427) with each server having services for translating XML into the particular Applications Programming Interface (API) language required by the server and for translating its internal language into XML prior to transmission to another server. In a preferred embodiment, all of these servers are contained in a single tier 3 platform, and may communicate with each other directly without the necessity of changing the interfacing protocol format. The Interface Server 417 (also alternatively designated herein as the WDK), communicates through a web server 405 via the internet 403 to web clients 401 via the **HTML** protocol. The Interface Server 417, also may communicate to a directly connected client 407 via other protocols such as XSL/XSLT etc., and may communicate to Personal Data Assistants 411 such as cell phones or Palm Pilots.TM. or other such wireless devices using wireless protocols such as WAP/WML, etc. The Interface Server 417,...

Claim 4

Claim 4's "a client side application portion for presenting a view to a user of a Web browser, said view allowing user interactions with said view, where some of said interactions specify given tests to perform on an object-oriented application component; and" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 4's "a server side application portion for receiving indications of said user interactions with said client side application portion and, responsive to said indications,

performing said given tests on said component." is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 5

Claim 5's "The test client application of claim 4 wherein said object-oriented application component is an Enterprise JavaBean." is anticipated by Helgeson et al, col. 2, lines 1-6, where it recites:

> ...required to allow distributed objects to communicate with each other across either the client-server or server-server boundary has been provided by the **EnterpriseJavaBeans (EJB).**TM. component architecture.This new architectural system and related tools and systems are well documented and well known to those skilled in these arts.

Claim 6

Claim 6's "The test client application of claim 4 wherein said client side application is described using Hyper-Text Markup Language. and JavaScript." is anticipated by Helgeson et al, col. 11, lines 39-67; col. 12, lines 1-6, where it recites:

> Referring now to FIG. 4, the tier 3 applications server 307 is expanded in FIG. 4 to illustrate the Business Applications Platform 415 of the present invention. In FIG. 4, the Platform contains an Interface Server 417, an Information Server 419, an Interconnect Server 423 and a Business Server 421. All of these Servers 417, 419, 421 and 423 may physically resign on the same hardware platform (such as a UNIX box or a Microsoft.TM. NT.TM. platform), or each server may reside on a separate hardware box, or any combination of servers and hardware boxes. Each of the servers may have included a JAVA Virtual Machine.TM. and the related **runtime** support. The electronic communications between these servers may use the XML protocol (409, 425, 427) with each server having services for translating XML into the particular Applications Programming Interface (API) language required by the server and for translating its internal language into XML prior to transmission to another server. In a preferred embodiment, all of these servers are contained in a single tier 3 platform, and may communicate with each other directly without the necessity of changing the interfacing protocol format. The Interface Server 417 (also alternatively designated herein as the WDK), communicates through a web server 405 via the internet 403 to web clients 401 via the **HTML** protocol. The Interface Server 417, also may communicate to a directly connected client 407 via other protocols such as XSL/XSLT etc., and may communicate to Personal Data Assistants 411 such as cell phones or Palm Pilots.TM. or other such wireless devices using wireless protocols such as WAP/WML, etc. The Interface Server 417, contains mechanisms to manipulate various kinds of display style sheets, to generate and execute web links, to manage dynamic content generation and dynamic generation of **Javascript**, all of which is described in more detail below in the section on the **Interface Server**/WDK 417.

Claim 7

Claim 7's "The test client application of claim 6 wherein said client side application is further described using JavaScript." is anticipated by Helgeson et al, col. 11, lines 39-67; col. 12, lines 1-6, where it recites:

> Referring now to FIG. 4, the tier 3 applications server 307 is expanded in FIG. 4 to illustrate the Business Applications Platform 415 of the present invention. In FIG. 4, the Platform contains an Interface Server 417, an Information Server 419, an Interconnect Server 423 and a Business Server 421. All of these Servers 417, 419, 421 and 423 may physically resign on the same hardware platform (such as a UNIX box or a Microsoft.TM. NT.TM. platform), or each server may reside on a separate hardware box, or any combination of servers and hardware boxes. Each of the servers may have included a JAVA Virtual Machine.TM. and the related runtime support. The electronic communications between these servers may use the XML protocol (409, 425, 427) with each server having services for translating XML into the particular Applications Programming Interface (API) language required by the server and for translating its internal language into XML prior to transmission to another server. In a preferred embodiment, all of these

servers are contained in a single tier 3 platform, and may communicate with each other directly without the necessity of changing the interfacing protocol format. The Interface Server 417 (also alternatively designated herein as the WDK), communicates through a web server 405 via the internet 403 to web clients 401 via the HTML protocol. The Interface Server 417, also may communicate to a directly connected client 407 via other protocols such as XSL/XSLT etc., and may communicate to Personal Data Assistants 411 such as cell phones or Palm Pilots.TM. or other such wireless devices using wireless protocols such as WAP/WML, etc. The Interface Server 417, contains mechanisms to manipulate various kinds of display style sheets, to generate and execute web links, to manage dynamic content generation and dynamic generation of **Javascript**, all of which is described in more detail below in the section on the **Interface Server**/WDK 417.

Claim 8

Claim 8's "The test client application of claim 4 wherein said server side application is implemented as JavaServer Pages." is anticipated by Helgeson et al, col. 11, lines 39-67; col. 12, lines 1-6, where it recites:

Referring now to FIG. 4, the tier 3 applications server 307 is expanded in FIG. 4 to illustrate the Business Applications Platform 415 of the present invention. In FIG. 4, the Platform contains an Interface Server 417, an Information Server 419, an Interconnect Server 423 and a Business Server 421. All of these Servers 417, 419, 421 and 423 may physically resign on the same hardware platform (such as a UNIX box or a Microsoft.TM. NT.TM. platform), or each server may reside on a separate hardware box, or any combination of servers and hardware boxes. Each of the servers may have included a JAVA Virtual Machine.TM. and the related runtime support. The electronic communications between these servers may use the XML protocol (409, 425, 427) with each server having services for translating XML into the particular Applications Programming Interface (API) language required by the server and for translating its internal language into XML prior to transmission to another server. In a preferred embodiment, all of these servers are contained in a single tier 3 platform, and may communicate with each other directly without the necessity of changing the interfacing protocol format. The Interface Server 417 (also alternatively designated herein as the WDK), communicates through a web server 405 via the internet 403 to web clients 401 via the HTML protocol. The Interface Server 417, also may communicate to a directly connected client 407 via other protocols such as XSL/XSLT etc., and may communicate to Personal Data Assistants 411 such as cell phones or Palm Pilots.TM. or other such wireless devices using wireless protocols such as WAP/WML, etc. The Interface Server 417, contains mechanisms to manipulate various kinds of display style sheets, to generate and execute web links, to manage dynamic content generation and dynamic generation of Javascript, all of which is described in more detail below in the section on the **Interface Server**/WDK 417.

Claim 9

Claim 9's "providing a test client user interface to a workstation over an HTTP link, where said test client user interface is viewed through the use of a Web browser run on said workstation;" is anticipated by Helgeson et al, col. 56, lines 19-30, where it recites:

> 1. Determines the SabaSite based on the request. The SabaSite is identified as follows:
>
> a. Extract the servlet path information from the request object using the **HttpServletRequest API** (getservletpath( )).
>
> b. If the servlet path ends with a Web Content Server 800 specific string suffix, then the associated SabaSite name is determined by stripping of that suffix.
>
> c. If the servlet path does not end with the Web Content Server 800 specific string suffix, then the system default SabaSite name is retrieved using the SabaSite API.

Claim 9's "receiving a selection from said workstation, said selection identifying a given object, where said given object is a home interface or a remote interface of said Enterprise JavaBean;" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the

> use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 9's "receiving a request from said workstation,. where said request is a consequence of user interaction with said test client user interface and includes an indication of a test to perform on said given object;" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 9's "responsive to said request, performing said test on said given object to give a result; and" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems.TM. JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the **client-server** system architecture with the development of JavaBeans.TM. and Java servelet generation. More recently, the technology...

Claim 9's "sending a response to said workstation over said HTTP link, said response including an indication of said result to be displayed by said user interface." is anticipated by Helgeson et al, col. 11, lines 39-67, where it recites:

> Referring now to FIG. 4, the tier 3 applications server 307 is expanded in FIG. 4 to illustrate the Business Applications Platform 415 of the present invention. In FIG. 4, the Platform contains an Interface Server 417, an Information Server 419, an Interconnect Server 423 and a Business Server 421. All of these Servers 417, 419, 421 and 423 may physically resign on the same hardware platform (such as a UNIX box or a Microsoft.TM. NT.TM. platform), or each server may reside on a separate hardware box, or any combination of servers and hardware boxes. Each of the servers may have included a JAVA Virtual Machine.TM. and the related **runtime** support. The electronic communications between these servers may use the XML protocol (409, 425, 427) with each server having services for translating XML into the particular Applications Programming Interface (API) language required by the server and for translating its internal language into XML prior to transmission to another server. In a preferred embodiment, all of these servers are contained in a single tier 3 platform, and may communicate with each other directly without the necessity of changing the interfacing protocol format. The Interface Server 417 (also alternatively designated herein as the WDK), communicates through a web server 405 via the internet 403 to web clients 401 via the **HTML** protocol. The Interface Server 417, also may communicate to a directly connected client 407 via other protocols such as XSL/XSLT etc., and may communicate to Personal Data Assistants 411 such as cell phones or Palm Pilots.TM. or other such wireless devices using wireless protocols such as WAP/WML, etc. The Interface Server 417,...

Claim 10

Claim 10's "The method of claim 9 wherein said performing said test on said given object comprises **invoking a method** of said object." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> Another Java API is based on the industry-standard Enterprise JavaBean (EJB) model. This model has a notion of "entity beans" that provide the interface to specific business objects. Accordingly, the persistence framework provides a EJB-based abstract class, "SabaEntityBean" that implements the javax.ejb.EntityBean interface. The SabaEntityBean class provides default implementations of the following **methods**: ejbActivate( ), ejbPassivate( ), ejbRemove( ), setEntityContext( ), ejbCreate( ), ejbLoad( ), ejbStore( ), and unsetEntityContext( ). Implementations of the ejbLoad( ),

> ejbStore( ), ejbCreate, and ejbRemove( ) methods rely on the selection, update, insertion, and deletion statements declared as part of metadata (please refer to the discussion of the implementation of SabaObject's API). Other methods are implemented as empty stubs that can be overridden by a developer if desired.

Claim 11

Claim 11's "The method of claim 9 wherein said test client user interface further provides a view that allows said user of said workstation to browse **Enterprise JavaBeans** in a given **Java Naming and Directory Interface (JNDI)** namespace." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 12

Claim 12's "The method of claim 11 wherein said test client user interface further provides a view that allows said user of said workstation to specify a particular JNDI server on which to allow said user to browse." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties

> such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 13

Claim 13's "The method of claim 9 wherein said test client user interface further provides a view that allows said user of said workstation to specify a given Enterprise JavaBean." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide Java Naming and Directory Service (JNDI) access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the EJB server, somewhat analogous to the JDBC connect string for locating a database), and then using the context, **look up the home interface of the bean by its name**. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 14

Claim 14's "The method of claim 13 wherein said test client user interface further provides a view that allows said user of said workstation to review **home interface** objects of said given Enterprise JavaBean." is anticipated by Helgeson et al, col. 25, lines 44-52, where it recites:

> In addition to defining the bean class, to implement an EJB one also needs to define a corresponding remote interface, a **home interface**, and, for entity beans, a primary key class. The remote interface is the external world's view

> of the bean and is comprised of the business methods that the bean wishes to expose. The getters and setters for the bean's attributes are also exposed through the remote interface. The home interface declares the life-cycle methods, such as those for creating, removing, or finding beans.

Claim 15

Claim 15's "The method of claim 14 where said Enterprise JavaBeans may inherit objects from a set of hierarchically higher Enterprise JavaBeans and wherein said test client user interface further provides a view that allows said user of said workstation to specify a sub-set of said set of hierarchically higher Enterprise JavaBeans from which to display methods in said view that allows said user of said workstation to review said objects." is anticipated by Helgeson et al, col. 5, lines 33-41, where it recites:

> The system makes use of some third party modules which are described in more detail below also. The terminology as used and described in these references for object, class, **inheritance**, component, container, bean, JavaBean, EJB, etc., are well known in these arts and are used herein generally without definition except where a specific meaning is assigned to a term herein.

Claim 16

Claim 16's "provide a test client user interface to a workstation over an **HTTP** link, where said test client user interface is viewed through the use of a Web browser run on said workstation;" is anticipated by Helgeson et al, col. 55, lines 48-55, where it recites:

> When the Cocoon engine processes the **HTTP** request, it invokes the getDocument( ) method of the file producer registered with Cocoon. Web Content Server 800 uses a specific file producer (SabaProducerFromFile) to

> load the requested file. This file producer uses SabaSite properties to determine the location of the requested file. To register the Web Content Server 800 specific file producer, the following line is added to cocoon properties:...

Claim 16's "receive a selection from said workstation, said selection identifying a given object, where said given object is a home interface or a remote interface of said Enterprise JavaBean;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 16's "receive a request from said workstation, where said request is a consequence of user interaction with said test client user interface and includes an indication of a test to perform on said given object;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in

> the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 16's "perform said test on said given object to give a result, responsive to said request; and" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 16's "send a response to said workstation over said HTTP link, said response including an indication of said result to be displayed by said user interface." is anticipated by Helgeson et al, col. 55, lines 48-55, where it recites:

> When the Cocoon engine processes the **HTTP** request, it invokes the getDocument( ) method of the file producer registered with Cocoon. Web Content Server 800 uses a specific file producer (SabaProducerFromFile) to load the requested file. This file producer uses SabaSite properties to determine the location of the requested file. To register the Web Content Server 800 specific file producer, the following line is added to cocoon properties:...

Claim 17

Claim 17's "select a given object;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "select a given method of said given object;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "supply said given method with a parameter;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "request that said given method be invoked with said parameter; " is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "responsive to receiving said request , invoke said method with said parameter to give a result; and" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "present a further user interface to present said result to said user." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18

Claim 18's "present a user interface over a data link, said user interface allowing a user to:" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18's "browse a Java Naming and Directory Interface namespace;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction

would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18's "select a given object in said Java Naming and Directory Interface namespace; and" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18's "receive information regarding said given object." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up the home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 19

Claim 19's "The computer readable medium of claim 18, said user interface further allowing a user to test said given object." is anticipated by Helgeson et al, col. 36, lines 19-24, where it recites:

> In EJB Specification 1.0, the deployment descriptor is a text file with a somewhat different format. The deployment descriptor is generally created using a **GUI tool**, generally supplied by EJB Server vendors. Additional information on deployment descriptors can be obtained from EJB literature and tool manuals.

Conclusion

20. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

A. Singer et al (U.S. Patent Number 6,557,009 B1; dated 29 April 2003; class 707; subclass 104.1) discloses an environmental permit web portal with data validation capabilities.

B. Williams (U.S. Patent Number 6,591,272 B1; dated 08 July 2003; class 707; subclass 102) discloses a method and apparatus to make and transmit objects from a database on a server computer to a client computer.

C. Grucci et al (U.S. Patent Number 6,604,209 B1; dated 05 August 2003; class 714; subclass 038) discloses distributed component testing in an enterprise computer system.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Wilbert L. Starks, Jr. whose telephone number is (703) 305-0027.

Alternatively, inquiries may be directed to the following:

S. P. E. Anil Khatri	**(703) 305-0282**
After-final (FAX)	**(703) 746-7238**
Official (FAX)	**(703) 746-7239**
Non-Official/Draft (FAX)	**(703) 746-7240**

WLS

11 December 2003

Wilbert L. Starks, Jr.
Primary Examiner
Art Unit - 2121